1. 계정키 생성
   * Filename: my-robo-project-220706-58354bda6e32.json
   * 생성하여 C:\Users\LnY에 다운로드
2. 환경 설정
   * GOOGLE\_APPLICATION\_CREDENTIALS=C:\Users\LnY\my-robo-project-220706-58354bda6e32.json

<https://rpaframework.org/libraries/cloud_google/index.html>

|  |  |
| --- | --- |
| **RPA.Cloud.Google**  **Library scope:** | Global |

## Introduction

Google is a library for operating with Google API endpoints.

Usage requires the following steps:

* Create a GCP project (o)
* Enable approriate APIs
* Create credentials (OAuth or service account)
* Install rpaframework-google package (o)

**Google authentication**

Authentication for Google is set with A) service account JSON file which can be given to the library in three different ways or with B) OAuth2 token, which is used for OAuth authentication.

**A) Methods when using service account:**

* Method 1 as environment variables, GOOGLE\_APPLICATION\_CREDENTIALS with path to service account file.
* Method 2 as keyword parameter to Init Storage for example.
* Method 3 as Robocorp vault secret. The vault name and secret key name needs to be given in library init or with keyword Set Robocorp Vault. Secret value should contain JSON file contents.

Method 1. service account using environment variable

**\*\*\* Settings \*\*\***

**Library** **RPA.Cloud.Google**

**\*\*\* Tasks \*\*\***

**Init Google services**

*# NO parameters for Init Vision, expecting to get JSON*

*# with GOOGLE\_APPLICATION\_CREDENTIALS environment variable*

Init Vision

실행 결과 #1)

PermissionDenied: 403 Cloud Vision API has not been used in project 378358435926 before or it is disabled. Enable it by visiting <https://console.developers.google.com/apis/api/vision.googleapis.com/overview?project=378358435926> then retry. If you enabled this API recently, wait a few minutes for the action to propagate to our systems and retry. [links { description: "Google developers console API activation" url: "<https://console.developers.google.com/apis/api/vision.googleapis.com/overview?project=378358435926>" } , reason: "SERVICE\_DISABLED" domain: "googleapis.com" metadata { key: "consumer" value: "projects/378358435926" } metadata { key: "service" value: "vision.googleapis.com" } ]

실행 결과 #2 – 서비스 상태를 ‘사용’으로 설정 후 다시 시도)

성공적으로 수행

한글은 문제?

Method 2. service account with keyword parameter

**\*\*\* Settings \*\*\***

**Library** **RPA.Cloud.Google**

**\*\*\* Tasks \*\*\***

**Init Google services**

Init Speech To Text /path/to/service\_account.json

실행 결과 #1 – 앞에서 서비스 상태를 ‘사용’으로 이미 설정)

성공적으로 수행

한글은 문제?

Method 3. setting Robocorp Vault in the library init

**\*\*\* Settings \*\*\***

**Library** **RPA.Cloud.Google**

... vault\_name=googlecloud

... vault\_secret\_key=servicecreds

**\*\*\* Tasks \*\*\***

**Init Google services**

Init Storage

Method 3. setting Robocorp Vault with keyword

**\*\*\* Settings \*\*\***

**Library** **RPA.Cloud.Google**

**\*\*\* Tasks \*\*\***

**Init Google services**

Set Robocorp Vault vault\_name=googlecloud vault\_secret\_key=servicecreds

Init Storage use\_robocorp\_vault=${TRUE}

**B) Methods when using OAuth token:**

* Method 1 as keyword parameter token\_file to Init Storage for example.
* Method 2 as Robocorp vault secret. The vault name and secret key name needs to be given in library init or with keyword Set Robocorp Vault. Secret value should contain JSON file contents.

Method 1. The Google Apps Script and Google Drive services are authenticated using this method.

**\*\*\* Settings \*\*\***

**Library** **RPA.Cloud.Google**

**\*\*\* Variables \*\*\***

@{SCRIPT\_SCOPES} forms spreadsheets

**\*\*\* Tasks \*\*\***

**Init Google OAuth services**

Init Apps Script token\_file=oauth\_token ${SCRIPT\_SCOPES}

Method 2. setting Robocorp Vault in the library init

**\*\*\* Settings \*\*\***

**Library** **RPA.Cloud.Google**

... vault\_name=googlecloud

... vault\_secret\_key=oauth

... cloud\_auth\_type=token

**\*\*\* Tasks \*\*\***

**Init Google services**

Init Storage

**Creating and using OAuth token file**

The token file can be created using credentials.json by running command:

rpa-google-oauth --credentials <filepath> --service drive or rpa-google-oauth --credentials <filepath> --scopes drive.appdata,drive.file,drive.install

This will start web based authentication process, which outputs the token at the end. Token could be stored into Robocorp Vault.

Example Vault content.

**"googlecloud"**: {

**"oauth-token"**: "gANfd123321aabeedYsc"

}

Using the Vault.

**\*\*\* Keywords \*\*\***

**Set up Google Drive authentication**

Set Robocorp Vault vault\_name=googlecloud

... vault\_secret\_key=oauth-token

... cloud\_auth\_type=token

Init Drive

**Installation**

This library, RPA.Cloud.Google is available via **rpaframework-google** package.

Check the latest package version from [PyPI](https://pypi.org/project/rpaframework-google/).

**Examples**

**Robot Framework**

**\*\*\* Settings \*\*\***

**Library** **RPA.Cloud.Google**

**\*\*\* Variables \*\*\***

${SERVICE\_ACCOUNT} ${/}path${/}to${/}service\_account.json

${BUCKET\_NAME} testbucket12213123123

**\*\*\* Tasks \*\*\***

**Upload a file into a new storage bucket**

[**Setup**] Init Storage ${SERVICE\_ACCOUNT}

Create Storage Bucket ${BUCKET\_NAME}

Upload Storage File ${BUCKET\_NAME}

... ${/}path${/}to${/}file.pdf

... myfile.pdf

@{files} List Storage Files ${BUCKET\_NAME}

FOR ${file} IN @{files}

Log ${file}

END

**Python**

**from** **RPA.Cloud.Google** **import** Google

library = Google

service\_account = '/path/to/service\_account.json'

library.init\_vision(service\_account)

library.init\_text\_to\_speech(service\_account)

response = library.detect\_text('imagefile.png', 'result.json')

library.synthesize\_speech('I want this said aloud', target\_file='said.mp3')

## Importing

#### Arguments

service\_account

=None

<str><None>vault\_name

=None

<str><None>vault\_secret\_key

=None

<str><None>cloud\_auth\_type

=serviceaccount

<str>

#### Documentation

Library initialization

|  |  |
| --- | --- |
| **service\_account:** | |
|  | path to service account |
| **vault\_name:** | Robocorp vault name |
| **vault\_secret\_key:** | |
|  | Robocorp secret key |
| **cloud\_auth\_type:** | |
|  | "serviceaccount" or "token", defaults to "serviceaccount" |

<http://bcho.tistory.com/1075>

[**구글 클라우드 비젼 API 사용하기 (Google Cloud Vision API)**](http://bcho.tistory.com/1075)

[빅데이타](http://bcho.tistory.com/category/%EB%B9%85%EB%8D%B0%EC%9D%B4%ED%83%80) | 2016.03.04 14:49 | Posted by 조대협

**Google Cloud Vision API 사용하기**

구글 클라우드 비젼 API 사용하기

조대협 (http://bcho.tistory.com)

빅데이타와 머신러닝과 같은 기술이 요즘 인터넷을 매우고 있는 시대에, 구글이 얼마전 이미지 디텍션 (Image detection)이 가능한, Cloud Vision API라는 오픈 API를 발표하였다. 현재는 베타버전 상태인데, 호기심에 빠르게 한번 테스트를 해봤다.

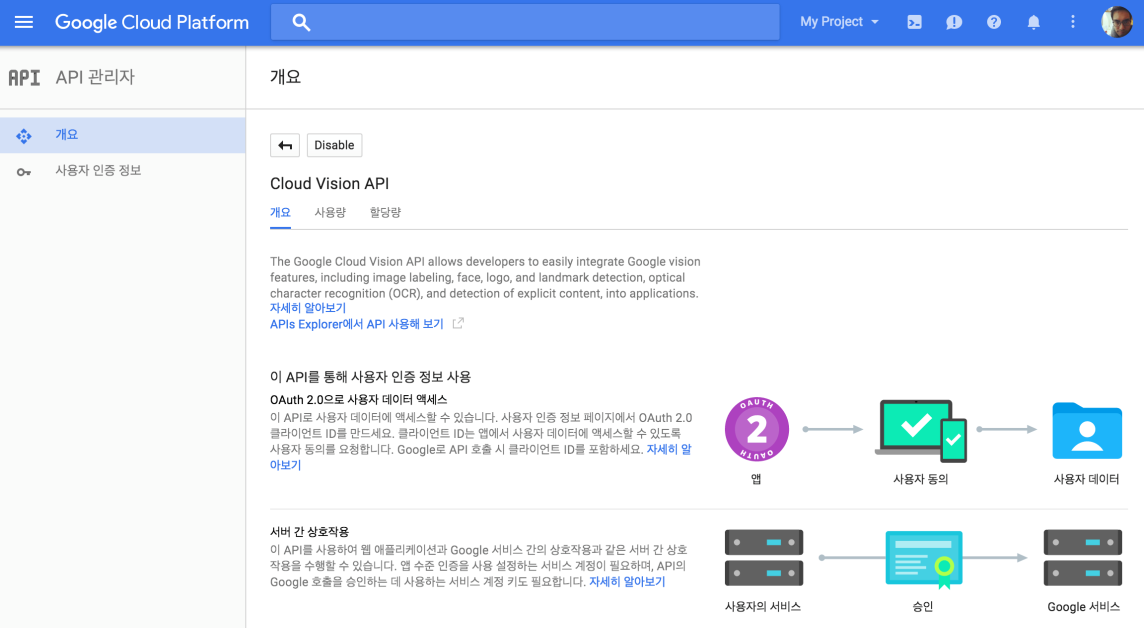
node.js를 이용하여, 간단한 테스트 프로그램을 만들어서 테스트를 해봤는데, 구현에 걸리는 시간은 불과 10분이 안된듯... (node.js는 역시 프로토타이핑용으로는 정말 좋은듯)

**Cloud Vision API 억세스 권한 얻기**

Cloud Vision API는 현재 베타 상태이다. 접근을 하려면 별도로 요청을 해야 접근 권한을 받을 수 있다. [**https://cloud.google.com/vision/**](https://cloud.google.com/vision/) 에서 권한을 신청하면 심사를 하고 권한을 준다.

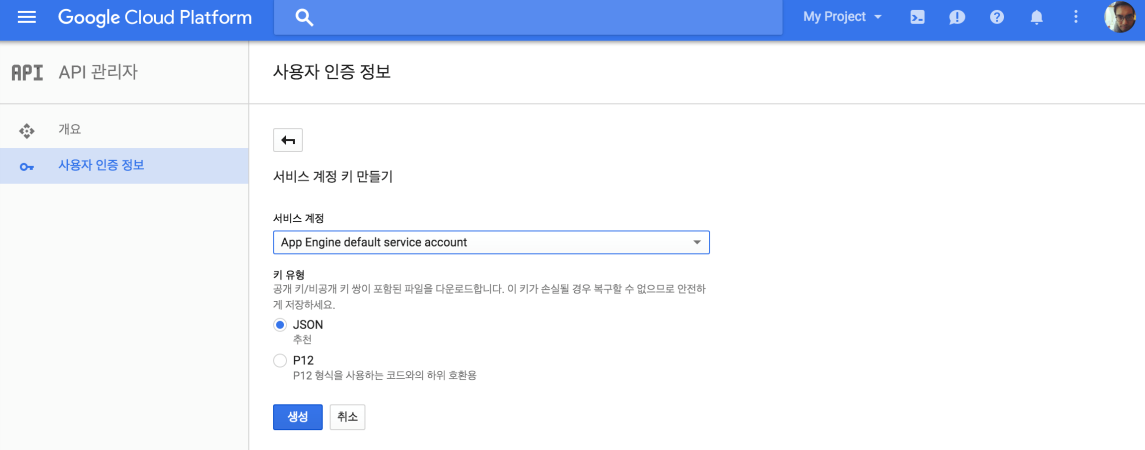
**Cloud Vision API 활성화 하기**

권한을 얻은 후에는 Google Cloud Platform API 관리자 콘솔에 접속해서 Cloud Vision API 사용을 활성화해야 한다.



다음으로 API를 외부에서 호출하기 위한 API 키를 발급받아야 하는데, Google Cloud Platform API에서는 API 키에 대한 다양한 접근 방식을 제공한다. OAuth 방식의 접근, 서버를 위한 API 키 발급 방식등 여러 방식을 지원하는데, 이 예제에서는 서비스 계정키를 JSON 형태로 다운 받아서 사용하도록 한다.

계정키를 생성 하는 방법은 Google Cloud Platform 콘솔에서 “사용자 인증 정보” 메뉴에서 “서비스 계정 키 만들기” 메뉴를 통해서 키를 생성하고 JSON 파일을 다운로드 받을 수 있다.



**Cloud Vision API 호출 하기**

<https://cloud.google.com/vision/docs/getting-started?utm_source=product-announcement&utm_medium=email&utm_campaign=2016-02-Vision-API&utm_content=NoFT>

를 보면 Cloud Vision API를 호출하는 방법이 자세하게 설명되어 있다. 제공하는 기능에 비해서 API 사용법은 무지 간단한데, JSON으로 REST 방식으로 API를 호출하면 분석 결과를 JSON으로 리턴 해준다. 이때 이미지 파일은 이 JSON에 base64 인코딩으로 첨부를 하면 된다. 어렵지 않으니 문서를 한번 쭈욱 보면서 흐름을 따라가보기를 권장한다.

**node.js 모듈 준비하기**

base64 인코딩 모듈을 넣어서 샘플 요청을 만드는 것 조차 귀찮아서 node.js의 npm모듈이 이미 있는지 찾아보기로 하였다. 역시나 있다. 테스트를 위해서 사용한 모듈은 google-vision-api-client 라는 모듈이다.<https://www.npmjs.com/package/google-vision-api-client>

%npm install google-vision-api-client

명령을 사용해서 모듈을 설치한다. 설치가 끝난후에, 모듈이 설치된 디렉토리 (내 경우에는 “/Users/terry/node\_modules/google-visionapi-client” - Mac OS)에 들어가서 index.js 파일을 열어보자. 이 모듈은 Cloud Vision API가 예전 알파 버전이었을 때 개발된 후 업데이트가 되지 않아서 현재 베타 버전의 Cloud Vision API호출이 안된다. Cloud Vision API의 End point가 변경되었기 때문인데.

index.js 파일에서 baseurl의 값을 다음과 같이 바꿔주자. (Cloud vision API의 베타 버전 URL로 변경)

var baseurl = ‘<https://vision.googleapis.com/v1/images:annotate'>;

이제 Cloud Vision API를 호출하기 위한 준비가 끝났다.

**node.js로 Cloud Vision API 호출 하기**

이제 google-vision-api-client를 이용하여 API를 호출해보자.다음은 google-vision-api-client에서 제공하는 예제이다.

var vision = require('google-vision-api-client');

var requtil = vision.requtil;

//Prepare your service account from trust preview certificated project

var jsonfile = '/Users/terry/dev/ws/nodejs/GoogleVisionAPISample/My Project-eee0a2d4532a.json';

//Initialize the api

vision.init(jsonfile);

//Build the request payloads

var d = requtil.createRequests().addRequest(

requtil.createRequest('/Users/terry/images/dale2.jpg')

.withFeature('FACE\_DETECTION', 3)

.withFeature('LABEL\_DETECTION', 2)

.build());

//Do query to the api server

vision.query(d, function(e, r, d){

if(e) console.log('ERROR:', e);

  console.log(JSON.stringify(d));

});

<예제. visionAPI.js >

코드를 작성하고, jsonfile 경로에 앞에서 다운받은 서비스 계정키 JSON 파일의 경로를 적어주면 된다.

그리고, createRequest 부분에, Google Cloud Vision API로 분석하고자 하는 이미지 파일명을 적고, withFeature라는 메서드를 이용해서 어떤 분석을 할것인지를 명시한다. (이 부분은 뒤에서 다시 설명한다.)

그러면 다음 명령을 통해서 실행을 해보자

%node visionAPI.js

실행을 하면 실행 결과를 json 형태로 리턴을 해주는데,

테스트에서 사용한 이미지와 결과는 다음과 같다.



<그림. 테스트에서 사용한 이미지 >

{

   "responses":[

      {

         "faceAnnotations":[

            {

               "boundingPoly":{

                  "vertices":[

                     {

                        "x":122,

                        "y":52

                     },

                     {

                        "x":674,

                        "y":52

                     },

                     {

                        "x":674,

                        "y":693

                     },

                     {

                        "x":122,

                        "y":693

                     }

                  ]

               },

               "fdBoundingPoly":{

                  "vertices":[

                     {

                        "x":176,

                        "y":208

                     },

                             중략...

                     }

                  ]

               },

               "landmarks":[

                  {

                     "type":"LEFT\_EYE",

                     "position":{

                        "x":282.99844,

                        "y":351.67017,

                        "z":-0.0033840234

                     }

                  },

                  {

                     "type":"RIGHT\_EYE",

                     "position":{

                        "x":443.8624,

                        "y":336.31445,

                        "z":-35.029751

                     }

                  },

               중략...

                  }

               ],

               "rollAngle":-3.8402841,

               "panAngle":-12.196975,

               "tiltAngle":-0.68598062,

               "detectionConfidence":0.8096019,

               "landmarkingConfidence":0.64295566,

               "joyLikelihood":"LIKELY",

               "sorrowLikelihood":"VERY\_UNLIKELY",

               "angerLikelihood":"VERY\_UNLIKELY",

               "surpriseLikelihood":"VERY\_UNLIKELY",

               "underExposedLikelihood":"VERY\_UNLIKELY",

               "blurredLikelihood":"VERY\_UNLIKELY",

               "headwearLikelihood":"VERY\_UNLIKELY"

            }

         ],

         "labelAnnotations":[

            {

               "mid":"/m/068jd",

               "description":"photograph",

               "score":0.92346138

            },

            {

               "mid":"/m/09jwl",

               "description":"musician",

               "score":0.86925673

            }

         ]

      }

   ]

}

<그림. 결과 JSON>

결과를 살펴보면, 눈코입의 위치와, 감정상태등을 상세하게 리턴해주는 것을 볼 수 있다.

joyLikeihood 는 기쁨 감정, sorrowLikeihood는 슬픈 감정들을 나타내는데,

Cloud Vision API는 여러개의 Feature를 동시에 분석이 가능하다.

예를 들어 얼굴 인식과 로고 인식을 같이 활용하여, 특정 브랜드를 보고 있을때 사람들의 얼굴 표정이 어떤지를 분석함으로써 대략적인 브랜드에 대한 반응을 인식한다던지. 특정 랜드마크와 표정 분석을 통해서 장소에 대한 분석 (재미있는 곳인지? 슬픈 곳인지.) 등으로 활용이 가능하다.

**Cloud Vision API의 이미지 분석 기능**

Cloud Vision API는 여러가지 형태의 이미지 분석 기능을 제공하는데, 대략적인 내용을 훝어보면 다음과 같다.

* Label Detection – 사진 속에 있는 사물을 찾아준다. 가구, 동물, 음식 등을 인지해서 리턴 해준다.
* Logo Detection – 사진 속에서 회사 로고와 같은 로고를 찾아준다.
* Landmark Detection – 사진 속에서 유명한 랜드 마크 (남산 타워, 경복궁등과 같은 건축물이나 자연 경관 이름)를 찾아준다.
* Face Detection – 사진 속에서 사람 얼굴을 찾아준다. 이게 좀 재미있는데 눈코입의 위치 등을 리턴하는 것을 물론 표정을 분석하여 감정 상태를 분석하여 리턴 해준다. 화가 났는지 기쁜 상태인지 슬픈 상태인지
* Safe Search Detection - 사진 컨텐츠의 위험도(? 또는 건전성)을 검출해주는데, 성인 컨텐츠, 의학 컨텐츠, 폭력 컨텐츠 등의 정도를 검출해준다.
* Optical Character Recognition - 문자 인식

그 외에도 몇 가지 추가적인 Feature를 제공하고 있으니 <https://cloud.google.com/vision/> 문서를 참고하기 바란다.

**Cloud Vision API는 무슨 의미를 제공하는가?**

사실 Cloud Vision API 자체로만으로도 대단히 흥미로운 기능을 가지고 있지만, Cloud Vision API는 또 다른 의미를 가지고 있다고 본다.

머신러닝이나 빅데이타, 그리고 인공 지능은 데이타 과학자와 대규모 하드웨어 자원을 가지고 있는 업체의 전유물이었지만, 근래에는 이러한 빅데이타 관련 기술들이 클라우드를 기반으로 하여 API로 제공됨으로써, 누구나 쉽게 빅데이타 기반의 분석 기술을 쉽게 활용할 수 있는 시대가 되어가고 있다.

이미 구글이나 Microsoft의 경우 머신러닝 알고리즘을 클라우드 API로 제공하고 있고, 대규모 데이타 분석의 경우에도 Google Analytics나 Yahoo Flurry등을 통해서 거의 무료로 제공이 되고 있다. (코드 몇 줄이면 앱에 추가도 가능하다.)

이러한 접근성을 통해서 많은 서비스와 앱 들이 고급 데이터 분석 알고리즘과 인공지능 기능들을 사용할 수 있는 보편화의 시대에 들어 선 것이 아닐까?